Private Arbitrated Loop Self-Test Management for a
Fibre Channel Storage Enclosure

Technical Field

5   The present invention relates to multi-peripheral-device enclosures, and, in particular, to a method and system for increasing the reliability and availability of a multi-peripheral-device enclosure by incorporating control elements for isolating components into the multi-peripheral-device enclosure so that the multi-peripheral-device enclosure can test the peripheral devices within the multi-

10   peripheral-device enclosure in order to identify malfunctioning peripheral devices and isolate the malfunctioning peripheral devices.

Background Of The Invention

           The fibre channel ("FC") is an architecture and protocol for a data

15   communication network for interconnecting a number of different combinations of computers and peripheral devices. The FC supports a variety of upper-level protocols, including the small computer systems interface ("SCSI") protocol. A computer or peripheral device is linked to the network through an FC port and copper wires or optical fibres. An FC port includes a transceiver and an interface

20   controller, and the computer peripheral device in which the FC port is contained is called a "host." The FC port exchanges data with the host via a local data bus, such as a peripheral computer interface ("PCI") bus. The interface controller conducts lower-level protocol exchanges between the fibre channel and the computer or peripheral device in which the FC port resides.

25   Because of the high bandwidth and flexible connectivity provided by the FC, the FC is becoming a common medium for interconnecting peripheral devices within multi-peripheral-device enclosures, such as redundant arrays of inexpensive disks ("RAIDs"), and for connecting multi-peripheral-device enclosures with one or more host computers. These multi-peripheral-device

enclosures economically provide greatly increased storage capacities and built-in redundancy that facilitates mirroring and fail over strategies needed in high-availability systems. Although the FC is well-suited for this application with regard to capacity and connectivity, the FC is a serial communications medium.

5 Malfunctioning peripheral devices and enclosures can, in certain cases, degrade or disable communications. A need has therefore been recognized for methods to improve the ability of FC-based multi-peripheral-device enclosures to isolate and recover from malfunctioning peripheral devices, and for improving the ability of systems including one or more host computers and multiple, interconnected FC-

10 based multi-peripheral-device enclosures to isolate and recover from a malfunctioning multi-peripheral-device enclosure. A need has also been recognized for additional communications and component redundancies within multi-peripheral-device enclosures to facilitate higher levels of fault-tolerance and high-availability.

15

Summary Of The Invention

The present invention provides a method and system for isolating peripheral devices within a multi-peripheral-device enclosure from the communications medium used to interconnect the peripheral devices within the

20 multi-peripheral-device enclosure, and for isolating a multi-peripheral-device enclosure from a communications medium used to interconnect a number of multi-peripheral-device enclosures with a host computer. The present invention provides increased component redundancy within multi-peripheral-device enclosures to eliminate single points of failure to increase fault-tolerance and high-availability of

25 the multi-peripheral-device enclosures.

Port bypass circuits are used to control access of peripheral devices to the communications medium used to interconnect the peripheral devices within the multi-peripheral-device enclosure. The port bypass circuits are themselves controlled by port bypass circuit controllers that can, in turn, be controlled by

software or firmware routines running on a microprocessor within the multi-peripheral-device enclosure. These three levels of control facilitate intelligent management of peripheral devices, diagnosis of malfunctioning peripheral devices, and isolation of malfunctioning peripheral devices. The three-tiered port bypass

5   circuit control is also extended to inter-multi-peripheral-device-enclosure connection ports, so that a malfunctioning multi-peripheral-device enclosure can be diagnosed and isolated from a communications medium connection the multi-peripheral-device enclosure to a host computer. Redundant port bypass circuit controllers and microprocessors can be used to improve reliability of the diagnosis

10 and isolation strategies implemented using the three-tiered port bypass circuit control.

The present invention provides a method by which a multi-peripheral-device enclosure can, upon being powered up, isolate itself from external host computers and from other, external multi-peripheral-device enclosures

15 in order to test the multi-peripheral-device enclosure's internal communications medium and to test each peripheral device within the multi-peripheral-device enclosure. Any peripheral devices found to be defective are bypassed via port bypass circuit controllers and port bypass circuits. If the internal communications medium is found to be defective, the method of the present invention can elect to

20 prevent the multi-peripheral-device enclosure from configuring itself into the communications medium that interconnects the multi-peripheral-device enclosure with external host computers and other, external multi-peripheral-device enclosures.

25 **Brief Description Of The Drawings**

FIGS. 1A-1C shows the three different types of FC interconnection topologies.

FIG. 2 illustrates a very simple hierarchy by which data is organized, in time, for transfer through an FC network.

FIG. 3 shows the contents of a standard FC frame.

FIG. 4 is a block diagram of a common personal computer architecture including a SCSI bus.

FIG. 5 illustrates the SCSI bus topology.

FIGS. 6A-6C illustrate the SCSI protocol involved in the initiation and implementation of read and write I/O operations.

FIGS. 7A and 7B illustrate a mapping of FCP sequences exchanged between an initiator and target and the SCSI bus phases and states described in FIGS. 6A-6C.

FIG. 8 shows a diagram of the seven phases of FC arbitrated loop initialization.

FIG. 9 shows the data payload of FC frames transmitted by FC nodes in an arbitrated loop topology during each of the seven phases of loop initialization shown in FIG. 9.

Fig. 10 illustrates a simple multi-peripheral devices enclosure.

Fig. 11 illustrates the basic communications paradigm represented by the SES command set.

Fig. 12 is a simplified illustration of the design used by manufacturers of certain currently-available FC-based multi-disk enclosures.

Fig. 13A is a schematic representation of a port bypass circuit, such as port bypass circuits 1222-1229 in Fig. 12.

Fig. 13B illustrates the connection of a disk drive to a fibre channel loop via a port bypass circuit.

Fig. 14 shows a highly available enclosure that incorporates techniques related to the present invention.

Fig. 15A illustrates control of a port bypass circuit by a port bypass circuit control chip.

Fig. 15B shows an example of the PBC control circuit implemented in hardware.

Figs. 16A-B illustrate the usefulness of implementing a shunting operation in order to bypass a GBIC.

Detailed Description Of The Invention

5   The present invention will be described below in six subsections. The first three subsections provide greater detail about the fibre channel architecture and protocol, the SCSI architecture and protocol, and implementation of the SCSI protocol on top of the fibre channel protocol. The fourth subsection discusses the fibre channel arbitrated loop intialization process. The fifth

10   subsection provides a general description of multi-peripheral-device enclosures, and the sixth subsection describes a specialized SCSI command set and protocol used for component management within systems of peripheral devices that communicate with one or more host computers via the SCSI protocol. The seventh subsection provides a detailed description of a hardware embodiment of the present invention,

15   and a final eighth subsection provides a pseudo-code implementation of an embodiment of the multi-peripheral-device enclosure self-test method.

Fibre Channel

        The Fibre Channel ("FC") is defined by, and described in, a number

20   of ANSI Standards documents, including: (1) Fibre Channel Physical and Signaling Interface ("FC-PH"), ANSI X3.230-1994, ("FC-PH-2), ANSI X3.297-1997; (2) Fibre Channel – Arbitrated Loop ("FC-AL-2"), ANSI X3.272-1996; (3) Fibre Channel – Private Loop SCSI Direct Attached ("FC-PLDA"); (4) Fibre Channel – Fabric Loop Attachment ("FC-FLA"); (5) Fibre Channel Protocol for SCSI

25   ("FCP"); (6) Fibre Channel Fabric Requirements ("FC-FG"), ANSI X3.289:1996; and (7) Fibre Channel 10-Bit Interface. These standards documents are under frequent revision. Additional Fibre Channel System Initiative ("FCSI") standards documents include: (1) Gigabaud Link Module Family ("GLM"), FCSI-301; (2) Common FC-PH Feature Sets Profiles, FCSI-101; and (3) SCSI Profile,

FCSI-201. These documents may be found at the world wide web Internet page having the following address:

"http://www.fibrechannel.com"

The following description of the FC is meant to introduce and summarize certain of the information contained in these documents in order to facilitate discussion of the present invention. If a more detailed discussion of any of the topics introduced in the following description is desired, the above-mentioned documents may be consulted.

The FC is an architecture and protocol for data communications between FC nodes, generally computers, workstations, peripheral devices, and arrays or collections of peripheral devices, such as disk arrays, interconnected by one or more communications media. Communications media include shielded twisted pair connections, coaxial cable, and optical fibers. An FC node is connected to a communications medium via at least one FC port and FC link. An FC port is an FC host adapter or FC controller that shares a register and memory interface with the processing components of the FC node, and that implements, in hardware and firmware, the lower levels of the FC protocol. The FC node generally exchanges data and control information with the FC port using shared data structures in shared memory and using control registers in the FC port. The FC port includes serial transmitter and receiver components coupled to a communications medium via a link that comprises electrical wires or optical strands.

In the following discussion, "FC" is used as an adjective to refer to the general Fibre Channel architecture and protocol, and is used as a noun to refer to an instance of a Fibre Channel communications medium. Thus, an FC (architecture and protocol) port may receive an FC (architecture and protocol) sequence from the FC (communications medium).

The FC architecture and protocol support three different types of interconnection topologies, shown in FIGS. 1A-1C. FIG. 1A shows the simplest of

the three interconnected topologies, called the "point-to-point topology." In the point-to-point topology shown in FIG. 1A, a first node 101 is directly connected to a second node 102 by directly coupling the transmitter 103 of the FC port 104 of the first node 101 to the receiver 105 of the FC port 106 of the second node 102,

5   and by directly connecting the transmitter 107 of the FC port 106 of the second node 102 to the receiver 108 of the FC port 104 of the first node 101. The ports 104 and 106 used in the point-to-point topology are called N_Ports.

FIG. 1B shows a somewhat more complex topology called the "FC arbitrated loop topology." FIG. 1B shows four nodes 110-113 interconnected

10   within an arbitrated loop. Signals, consisting of electrical or optical binary data, are transferred from one node to the next node around the loop in a circular fashion. The transmitter of one node, such as transmitter 114 associated with node 111, is directly connected to the receiver of the next node in the loop, in the case of transmitter 114, with the receiver 115 associated with node 112. Two types

15   of FC ports may be used to interconnect FC nodes within an arbitrated loop. The most common type of port used in arbitrated loops is called the "NL_Port." A special type of port, called the "FL_Port," may be used to interconnect an FC arbitrated loop with an FC fabric topology, to be described below. Only one FL_Port may be actively incorporated into an arbitrated loop topology. An FC

20   arbitrated loop topology may include up to 127 active FC ports, and may include additional non-participating FC ports.

In the FC arbitrated loop topology, nodes contend for, or arbitrate for, control of the arbitrated loop. In general, the node with the lowest port address obtains control in the case that more than one node is contending for

25   control. A fairness algorithm may be implemented by nodes to ensure that all nodes eventually receive control within a reasonable amount of time. When a node has acquired control of the loop, the node can open a channel to any other node within the arbitrated loop. In a half duplex channel, one node transmits and the other node receives data. In a full duplex channel, data may be transmitted by a

first node and received by a second node at the same time that data is transmitted

by the second node and received by the first node. For example, if, in the

arbitrated loop of FIG. 1B, node 111 opens a full duplex channel with node 113,

then data transmitted through that channel from node 111 to node 113 passes

5   through NL_Port 116 of node 112, and data transmitted by node 113 to node 111

passes through NL_Port 117 of node 110.

FIG. 1C shows the most general and most complex FC topology,

called an "FC fabric." The FC fabric is represented in FIG. 1C by the irregularly

shaped central object 118 to which four FC nodes 119-122 are connected. The

10   N_Ports 123-126 within the FC nodes 119-122 are connected to F_Ports 127-130

within the fabric 118. The fabric is a switched or cross-point switch topology

similar in function to a telephone system. Data is routed by the fabric between

F_Ports through switches or exchanges called "fabric elements." There may be

many possible routes through the fabric between one F_Port and another F_Port.

15   The routing of data and the addressing of nodes within the fabric associated with

F_Ports are handled by the FC fabric, rather than by FC nodes or N_Ports.

When optical fibers are employed, a single FC fabric can extend for

ten kilometers. The FC can support interconnection of more than 16,000,000 FC

nodes. A single FC host adapter can transmit and receive data at rates of up to 200

20   Mbytes per second. Much higher data exchange rates are planned for FC

components in the near future.

The FC is a serial communications medium. Data is transferred one

bit at a time at extremely high transfer rates. FIG. 2 illustrates a very simple

hierarchy by which data is organized, in time, for transfer through an FC network.

25   At the lowest conceptual level, the data can be considered to be a stream of data

bits 200. The smallest unit of data, or grouping of data bits, supported by an FC

network is a 10-bit character that is decoded by FC port as an 8-bit character. FC

primitives are composed of 10-byte characters or bytes. Certain FC primitives are

employed to carry control information exchanged between FC ports. The next

level of data organization, a fundamental level with regard to the FC protocol, is a frame. Seven frames 202-208 are shown in FIG. 2. A frame may be composed of between 36 and 2,148 bytes of data, depending on the nature of the data included in the frame. The first FC frame, for example, corresponds to the data bits of the stream of data bits 200 encompassed by the horizontal bracket 201. The FC protocol specifies a next higher organizational level called the sequence. A first sequence 210 and a portion of a second sequence 212 are displayed in FIG. 2. The first sequence 210 is composed of frames one through four 202-205. The second sequence 212 is composed of frames five through seven 206-208 and additional frames that are not shown. The FC protocol specifies a third organizational level called the exchange. A portion of an exchange 214 is shown in FIG. 2. This exchange 214 is composed of at least the first sequence 210 and the second sequence 212 shown in FIG. 2. This exchange can alternatively be viewed as being composed of frames one through seven 202-208, and any additional frames contained in the second sequence 212 and in any additional sequences that compose the exchange 214.

The FC is a full duplex data transmission medium. Frames and sequences can be simultaneously passed in both directions between an originator, or initiator, and a responder, or target. An exchange comprises all sequences, and frames within the sequences, exchanged between an originator and a responder during a single I/O transaction, such as a read I/O transaction or a write I/O transaction. The FC protocol is designed to transfer data according to any number of higher-level data exchange protocols, including the Internet protocol ("IP"), the Small Computer Systems Interface ("SCSI") protocol, the High Performance Parallel Interface ("HIPPI"), and the Intelligent Peripheral Interface ("IPI"). The SCSI bus architecture will be discussed in the following subsection, and much of the subsequent discussion in this and remaining subsections will focus on the SCSI protocol embedded within the FC protocol. The standard adaptation of SCSI protocol to fibre channel is subsequently referred to in this document as "FCP.".

Thus, the FC can support a master-slave type communications paradigm that is characteristic of the SCSI bus and other peripheral interconnection buses, as well as the relatively open and unstructured communication protocols such as those used to implement the Internet. The SCSI bus architecture concepts of an initiator and

5   target are carried forward in the FCP, designed, as noted above, to encapsulate SCSI commands and data exchanges for transport through the FC.

FIG. 3 shows the contents of a standard FC frame. The FC frame 302 comprises five high level sections 304, 306, 308, 310 and 312. The first high level section, called the start-of-frame deliminator 304, comprises 4 bytes that

10   mark the beginning of the frame. The next high level section, called frame header 306, comprises 24 bytes that contain addressing information, sequence information, exchange information, and various control flags. A more detailed view of the frame header 314 is shown expanded from the FC frame 302 in FIG. 3. The destination identifier ("D_ID"), or DESTINATION_ID 316, is a 24-bit FC

15   address indicating the destination FC port for the frame. The source identifier ("S_ID"), or SOURCE_ID 318, is a 24-bit address that indicates the FC port that transmitted the frame. The originator ID, or OX_ID 320, and the responder ID 322, or RX_ID, together compose a 32-bit exchange ID that identifies the exchange to which the frame belongs with respect to the originator, or initiator, and

20   responder, or target, FC ports. The sequence ID, or SEQ_ID, 324 identifies the sequence to which the frame belongs.

The next high level section 308, called the data payload, contains the actual data packaged within the FC frame. The data payload contains data and encapsulating protocol information that is being transferred according to a higher-

25   level protocol, such as IP and SCSI. FIG. 3 shows four basic types of data payload layouts 326-329 used for data transfer according to the SCSI protocol. The first of these formats 326, called the FCP_CMND, is used to send a SCSI command from an initiator to a target. The FCP_LUN field 330 comprises an 8-byte address that may, in certain implementations, specify a particular SCSI-bus adapter, a target

device associated with that SCSI-bus adapter, and a logical unit number ("LUN") corresponding to a logical device associated with the specified target SCSI device that together represent the target for the FCP_CMND. In other implementations, the FCP_LUN field 330 contains an index or reference number that can be used by

5   the target FC host adapter to determine the SCSI-bus adapter, a target device associated with that SCSI-bus adapter, and a LUN corresponding to a logical device associated with the specified target SCSI device. An actual SCSI command, such as a SCSI read or write I/O command, is contained within the 16-byte field FCP_CDB 332.

10   The second type of data payload format 327 shown in FIG. 3 is called the FCP_XFER_RDY layout. This data payload format is used to transfer a SCSI proceed command from the target to the initiator when the target is prepared to begin receiving or sending data. The third type of data payload format 328 shown in FIG. 3 is the FCP_DATA format, used for transferring the actual data

15   that is being read or written as a result of execution of a SCSI I/O transaction. The final data payload format 329 shown in FIG. 3 is called the FCP_RSP layout, used to transfer a SCSI status byte 334, as well as other FCP status information, from the target back to the initiator upon completion of the I/O transaction.

20   **The SCSI Bus Architecture**

A computer bus is a set of electrical signal lines through which computer commands and data are transmitted between processing, storage, and input/output ("I/O") components of a computer system. The SCSI I/O bus is the most widespread and popular computer bus for interconnecting mass storage

25   devices, such as hard disks and CD-ROM drives, with the memory and processing components of computer systems. The SCSI bus architecture is defined in three major standards: SCSI-1, SCSI-2 and SCSI-3. The SCSI-1 and SCSI-2 standards are published in the American National Standards Institute ("ANSI") standards documents "X3.131-1986," and "X3.131-1994," respectively. The SCSI-3

standard is currently being developed by an ANSI committee. An overview of the SCSI bus architecture is provided by "The SCSI Bus and IDE Interface," Freidhelm Schmidt, Addison-Wesley Publishing Company, ISBN 0-201-17514-2, 1997 ("Schmidt").

5   FIG. 4 is a block diagram of a common personal computer ("PC") architecture including a SCSI bus. The PC 400 includes a central processing unit, or processor ("CPU") 402, linked to a system controller 404 by a high-speed CPU bus 406. The system controller is, in turn, linked to a system memory component 408 via a memory bus 410. The system controller 404 is, in addition,

10   linked to various peripheral devices via a peripheral component interconnect ("PCI") bus 412 that is interconnected with a slower industry standard architecture ("ISA") bus 414 and a SCSI bus 416. The architecture of the PCI bus is described in "PCI System Architecture," Shanley & Anderson, Mine Share, Inc., Addison-Wesley Publishing Company, ISBN 0-201-40993-3, 1995. The interconnected

15   CPU bus 406, memory bus 410, PCI bus 412, and ISA bus 414 allow the CPU to exchange data and commands with the various processing and memory components and I/O devices included in the computer system. Generally, very high-speed and high bandwidth I/O devices, such as a video display device 418, are directly connected to the PCI bus. Slow I/O devices 420, such as a keyboard 420 and a

20   pointing device (not shown), are connected directly to the ISA bus 414. The ISA bus is interconnected with the PCI bus through a bus bridge component 422. Mass storage devices, such as hard disks, floppy disk drives, CD-ROM drives, and tape drives 424-426 are connected to the SCSI bus 416. The SCSI bus is interconnected with the PCI bus 412 via a SCSI-bus adapter 430. The SCSI-bus adapter 430

25   includes a processor component, such as processor selected from the Symbios family of 53C8xx SCSI processors, and interfaces to the PCI bus 412 using standard PCI bus protocols. The SCSI-bus adapter 430 interfaces to the SCSI bus 416 using the SCSI bus protocol that will be described, in part, below. The SCSI-bus adapter 430 exchanges commands and data with SCSI controllers (not

shown) that are generally embedded within each mass storage device 424-426, or SCSI device, connected to the SCSI bus. The SCSI controller is a hardware/firmware component that interprets and responds to SCSI commands received from a SCSI adapter via the SCSI bus and that implements the SCSI

5   commands by interfacing with, and controlling, logical devices. A logical device may correspond to one or more physical devices, or to portions of one or more physical devices. Physical devices include data storage devices such as disk, tape and CD-ROM drives.

Two important types of commands, called I/O commands, direct the

10   SCSI device to read data from a logical device and write data to a logical device. An I/O transaction is the exchange of data between two components of the computer system, generally initiated by a processing component, such as the CPU 402, that is implemented, in part, by a read I/O command or by a write I/O command. Thus, I/O transactions include read I/O transactions and write I/O

15   transactions.

The SCSI bus 416 is a parallel bus that can simultaneously transport a number of data bits. The number of data bits that can be simultaneously transported by the SCSI bus is referred to as the width of the bus. Different types of SCSI buses have widths of 8, 16 and 32 bits. The 16 and 32-bit SCSI buses are

20   referred to as wide SCSI buses.

As with all computer buses and processors, the SCSI bus is controlled by a clock that determines the speed of operations and data transfer on the bus. SCSI buses vary in clock speed. The combination of the width of a SCSI bus and the clock rate at which the SCSI bus operates determines the number of

25   bytes that can be transported through the SCSI bus per second, or bandwidth of the SCSI bus. Different types of SCSI buses have bandwidths ranging from less than 2 megabytes ("Mbytes") per second up to 40 Mbytes per second, with increases to 80 Mbytes per second and possibly 160 Mbytes per second planned for the future.

The increasing bandwidths may be accompanied by increasing limitations in the physical length of the SCSI bus.

FIG. 5 illustrates the SCSI bus topology. A computer system 502, or other hardware system, may include one or more SCSI-bus adapters 504

5 and 506. The SCSI-bus adapter, the SCSI bus which the SCSI-bus adapter controls, and any peripheral devices attached to that SCSI bus together comprise a domain. SCSI-bus adapter 504 in FIG. 5 is associated with a first domain 508 and SCSI-bus adapter 506 is associated with a second domain 510. The most current SCSI-2 bus implementation allows fifteen different SCSI devices 513-515

10 and 516-517 to be attached to a single SCSI bus. In FIG. 5, SCSI devices 513-515 are attached to SCSI bus 518 controlled by SCSI-bus adapter 506, and SCSI devices 516-517 are attached to SCSI bus 520 controlled by SCSI-bus adapter 504. Each SCSI-bus adapter and SCSI device has a SCSI identification number, or SCSI_ID, that uniquely identifies the device or adapter in a particular SCSI bus.

15 By convention, the SCSI-bus adapter has SCSI_ID 7, and the SCSI devices attached to the SCSI bus have SCSI_IDs ranging from 0 to 6 and from 8 to 15. A SCSI device, such as SCSI device 513, may interface with a number of logical devices, each logical device comprising portions of one or more physical devices. Each logical device is identified by a logical unit number ("LUN") that uniquely

20 identifies the logical device with respect to the SCSI device that controls the logical device. For example, SCSI device 513 controls logical devices 522-524 having LUNs 0, 1, and 2, respectively. According to SCSI terminology, a device that initiates an I/O command on the SCSI bus is called an initiator, and a SCSI device that receives an I/O command over the SCSI bus that directs the SCSI device to

25 execute an I/O operation is called a target.

In general, a SCSI-bus adapter, such as SCSI-bus adapters 504 and 506, initiates I/O operations by sending commands to target devices. The target devices 513-515 and 516-517 receive the I/O commands from the SCSI bus. The target devices 513-515 and 516-517 then implement the commands by

interfacing with one or more logical devices that they control to either read data from the logical devices and return the data through the SCSI bus to the initiator or to write data received through the SCSI bus from the initiator to the logical devices. Finally, the target devices 513-515 and 516-517 respond to the initiator through the

5   SCSI bus with status messages that indicate the success or failure of implementation of the commands.

FIGS. 6A-6C illustrate the SCSI protocol involved in the initiation and implementation of read and write I/O operations. Read and write I/O operations compose the bulk of I/O operations performed by SCSI devices. Efforts

10   to maximize the efficiency of operation of a system of mass storage devices interconnected by a SCSI bus are most commonly directed toward maximizing the efficiency at which read and write I/O operations are performed. Thus, in the discussions to follow, the architectural features of various hardware devices will be discussed in terms of read and write operations.

15   FIG. 6A shows the sending of a read or write I/O command by a SCSI initiator, most commonly a SCSI-bus adapter, to a SCSI target, most commonly a SCSI controller embedded in a SCSI device associated with one or more logical devices. The sending of a read or write I/O command is called the command phase of a SCSI I/O operation. FIG. 6A is divided into initiator 602 and

20   target 604 sections by a central vertical line 606. Both the initiator and the target sections include columns entitled "state" 606 and 608 that describe the state of the SCSI bus and columns entitled "events" 610 and 612 that describe the SCSI bus events associated with the initiator and the target, respectively. The bus states and bus events involved in the sending of the I/O command are ordered in time,

25   descending from the top of FIG. 6A to the bottom of FIG 6A. FIGS. 6B-6C also adhere to this above-described format.

The sending of an I/O command from an initiator SCSI-bus adapter to a target SCSI device, illustrated in FIG. 6A, initiates a read or write I/O operation by the target SCSI device. Referring to FIG. 4, the SCSI-bus

adapter 430 initiates the I/O operation as part of an I/O transaction. Generally, the SCSI-bus adapter 430 receives a read or write command via the PCI bus 412, system controller 404, and CPU bus 406, from the CPU 402 directing the SCSI-bus adapter to perform either a read operation or a write operation. In a read

5   operation, the CPU 402 directs the SCSI-bus adapter 430 to read data from a mass storage device 424-426 and transfer that data via the SCSI bus 416, PCI bus 412, system controller 404, and memory bus 410 to a location within the system memory 408. In a write operation, the CPU 402 directs the system controller 404 to transfer data from the system memory 408 via the memory bus 410, system

10 controller 404, and PCI bus 412 to the SCSI-bus adapter 430, and directs the SCSI-bus adapter 430 to send the data via the SCSI bus 416 to a mass storage device 424-426 on which the data is written.

FIG. 6A starts with the SCSI bus in the BUS FREE state 614, indicating that there are no commands or data currently being transported on the

15 SCSI device. The initiator, or SCSI-bus adapter, asserts the BSY, D7 and SEL signal lines of the SCSI bus in order to cause the bus to enter the ARBITRATION state 616. In this state, the initiator announces to all of the devices an intent to transmit a command on the SCSI bus. Arbitration is necessary because only one device may control operation of the SCSI bus at any instant in time. Assuming that

20 the initiator gains control of the SCSI bus, the initiator then asserts the ATN signal line and the DX signal line corresponding to the target SCSI_ID in order to cause the SCSI bus to enter the SELECTION state 618. The initiator or target asserts and drops various SCSI signal lines in a particular sequence in order to effect a SCSI bus state change, such as the change of state from the ARBITRATION state 616 to

25 the SELECTION state 618, described above. These sequences can be found in Schmidt and in the ANSI standards, and will therefore not be further described below.

When the target senses that the target has been selected by the initiator, the target assumes control 620 of the SCSI bus in order to complete the

command phase of the I/O operation. The target then controls the SCSI signal lines in order to enter the MESSAGE OUT state 622. In a first event that occurs in the MESSAGE OUT state, the target receives from the initiator an IDENTIFY message 623. The IDENTIFY message 623 contains a LUN field 624 that

5   identifies the LUN to which the command message that will follow is addressed. The IDENTIFY message 623 also contains a flag 625 that is generally set to indicate to the target that the target is authorized to disconnect from the SCSI bus during the target's implementation of the I/O command that will follow. The target then receives a QUEUE TAG message 626 that indicates to the target how the I/O

10   command that will follow should be queued, as well as providing the target with a queue tag 627. The queue tag is a byte that identifies the I/O command. A SCSI-bus adapter can therefore concurrently manage 656 different I/O commands per LUN. The combination of the SCSI_ID of the initiator SCSI-bus adapter, the SCSI_ID of the target SCSI device, the target LUN, and the queue tag together

15   comprise an I_T_L_Q nexus reference number that uniquely identifies the I/O operation corresponding to the I/O command that will follow within the SCSI bus. Next, the target device controls the SCSI bus signal lines in order to enter the COMMAND state 628. In the COMMAND state, the target solicits and receives from the initiator the I/O command 630. The I/O command 630 includes an

20   opcode 632 that identifies the particular command to be executed, in this case a read command or a write command, a logical block number 636 that identifies the logical block of the logical device that will be the beginning point of the read or write operation specified by the command, and a data length 638 that specifies the number of blocks that will be read or written during execution of the command.

25   When the target has received and processed the I/O command, the target device controls the SCSI bus signal lines in order to enter the MESSAGE IN state 640 in which the target device generally sends a disconnect message 642 back to the initiator device. The target disconnects from the SCSI bus because, in general, the target will begin to interact with the logical device in order to prepare

the logical device for the read or write operation specified by the command. The target may need to prepare buffers for receiving data, and, in the case of disk drives or CD-ROM drives, the target device may direct the logical device to seek to the appropriate block specified as the starting point for the read or write command.

5   By disconnecting, the target device frees up the SCSI bus for transportation of additional messages, commands, or data between the SCSI-bus adapter and the target devices. In this way, a large number of different I/O operations can be concurrently multiplexed over the SCSI bus. Finally, the target device drops the BSY signal line in order to return the SCSI bus to the BUS FREE state 644.

10   The target device then prepares the logical device for the read or write operation. When the logical device is ready for reading or writing data, the data phase for the I/O operation ensues. FIG. 6B illustrates the data phase of a SCSI I/O operation. The SCSI bus is initially in the BUS FREE state 646. The target device, now ready to either return data in response to a read I/O command or

15   accept data in response to a write I/O command, controls the SCSI bus signal lines in order to enter the ARBITRATION state 648. Assuming that the target device is successful in arbitrating for control of the SCSI bus, the target device controls the SCSI bus signal lines in order to enter the RESELECTION state 650. The RESELECTION state is similar to the SELECTION state, described in the above

20   discussion of FIG. 6A, except that it is the target device that is making the selection of a SCSI-bus adapter with which to communicate in the RESELECTION state, rather than the SCSI-bus adapter selecting a target device in the SELECTION state.

        Once the target device has selected the SCSI-bus adapter, the target device manipulates the SCSI bus signal lines in order to cause the SCSI bus to enter

25   the MESSAGE IN state 652. In the MESSAGE IN state, the target device sends both an IDENTIFY message 654 and a QUEUE TAG message 656 to the SCSI-bus adapter. These messages are identical to the IDENTITY and QUEUE TAG messages sent by the initiator to the target device during transmission of the I/O command from the initiator to the target, illustrated in FIG. 6A. The initiator may

use the I_T_L_Q nexus reference number, a combination of the SCSI_IDs of the initiator and target device, the target LUN, and the queue tag contained in the QUEUE TAG message, to identify the I/O transaction for which data will be subsequently sent from the target to the initiator, in the case of a read operation, or

5   to which data will be subsequently transmitted by the initiator, in the case of a write operation. The I_T_L_Q nexus reference number is thus an I/O operation handle that can be used by the SCSI-bus adapter as an index into a table of outstanding I/O commands in order to locate the appropriate buffer for receiving data from the target device, in case of a read, or for transmitting data to the target device, in case

10 of a write.

After sending the IDENTIFY and QUEUE TAG messages, the target device controls the SCSI signal lines in order to transition to a DATA state 658. In the case of a read I/O operation, the SCSI bus will transition to the DATA IN state. In the case of a write I/O operation, the SCSI bus will transition

15 to a DATA OUT state. During the time that the SCSI bus is in the DATA state, the target device will transmit, during each SCSI bus clock cycle, a data unit having a size, in bits, equal to the width of the particular SCSI bus on which the data is being transmitted. In general, there is a SCSI bus signal line handshake involving the signal lines ACK and REQ as part of the transfer of each unit of data. In the

20 case of a read I/O command, for example, the target device places the next data unit on the SCSI bus and asserts the REQ signal line. The initiator senses assertion of the REQ signal line, retrieves the transmitted data from the SCSI bus, and asserts the ACK signal line to acknowledge receipt of the data. This type of data transfer is called asynchronous transfer. The SCSI bus protocol also allows for the

25 target device to transfer a certain number of data units prior to receiving the first acknowledgment from the initiator. In this transfer mode, called synchronous transfer, the latency between the sending of the first data unit and receipt of acknowledgment for that transmission is avoided. During data transmission, the target device can interrupt the data transmission by sending a SAVE POINTERS

message followed by a DISCONNECT message to the initiator and then controlling the SCSI bus signal lines to enter the BUS FREE state. This allows the target device to pause in order to interact with the logical devices which the target device controls before receiving or transmitting further data. After disconnecting from the

5   SCSI bus, the target device may then later again arbitrate for control of the SCSI bus and send additional IDENTIFY and QUEUE TAG messages to the initiator so that the initiator can resume data reception or transfer at the point that the initiator was interrupted. An example of disconnect and reconnect 660 are shown in FIG. 3B interrupting the DATA state 658. Finally, when all the data for the I/O

10   operation has been transmitted, the target device controls the SCSI signal lines in order to enter the MESSAGE IN state 662, in which the target device sends a DISCONNECT message to the initiator, optionally preceded by a SAVE POINTERS message. After sending the DISCONNECT message, the target device drops the BSY signal line so the SCSI bus transitions to the BUS FREE state 664.

15   Following the transmission of the data for the I/O operation, as illustrated in FIG. 6B, the target device returns a status to the initiator during the status phase of the I/O operation. FIG. 6C illustrates the status phase of the I/O operation. As in FIGS. 6A-6B, the SCSI bus transitions from the BUS FREE state 666 to the ARBITRATION state 668, RESELECTION state 670, and

20   MESSAGE IN state 672, as in FIG. 3B. Following transmission of an IDENTIFY message 674 and QUEUE TAG message 676 by the target to the initiator during the MESSAGE IN state 672, the target device controls the SCSI bus signal lines in order to enter the STATUS state 678. In the STATUS state 678, the target device sends a single status byte 684 to the initiator to indicate whether or not the I/O

25   command was successfully completed. In FIG. 6C, the status byte 680 corresponding to a successful completion, indicated by a status code of 0, is shown being sent from the target device to the initiator. Following transmission of the status byte, the target device then controls the SCSI bus signal lines in order to enter the MESSAGE IN state 682, in which the target device sends a COMMAND

COMPLETE message 684 to the initiator. At this point, the I/O operation has been completed. The target device then drops the BSY signal line so that the SCSI bus returns to the BUS FREE state 686. The SCSI-bus adapter can now finish its portion of the I/O command, free up any internal resources that were allocated in order to execute the command, and return a completion message or status back to the CPU via the PCI bus.

Mapping the SCSI Protocol onto FCP

FIGS. 7A and 7B illustrate a mapping of FCP sequences exchanged between an initiator and target and the SCSI bus phases and states described in FIGS. 6A-6C. In FIGS. 7A-7B, the target SCSI adapter is assumed to be packaged together with a FCP host adapter, so that the target SCSI adapter can communicate with the initiator via the FC and with a target SCSI device via the SCSI bus. FIG. 7A shows a mapping between FCP sequences and SCSI phases and states for a read I/O transaction. The transaction is initiated when the initiator sends a single-frame FCP sequence containing a FCP_CMND data payload through the FC to a target SCSI adapter 702. When the target SCSI-bus adapter receives the FCP_CMND frame, the target SCSI-bus adapter proceeds through the SCSI states of the command phase 704 illustrated in FIG. 6A, including ARBITRATION, RESELECTION, MESSAGE OUT, COMMAND, and MESSAGE IN. At the conclusion of the command phase, as illustrated in FIG. 6A, the SCSI device that is the target of the I/O transaction disconnects from the SCSI bus in order to free up the SCSI bus while the target SCSI device prepares to execute the transaction. Later, the target SCSI device rearbitrates for SCSI bus control and begins the data phase of the I/O transaction 706. At this point, the SCSI-bus adapter may send a FCP_XFER_RDY single-frame sequence 708 back to the initiator to indicate that data transmission can now proceed. In the case of a read I/O transaction, the FCP_XFER_RDY single-frame sequence is optional. As the data phase continues, the target SCSI device begins to read data from a logical device and transmit that data over the SCSI bus to the target SCSI-bus adapter. The target SCSI-bus adapter

then packages the data received from the target SCSI device into a number of FCP_DATA frames that together compose the third sequence of the exchange corresponding to the I/O read transaction, and transmits those FCP_DATA frames back to the initiator through the FC. When all the data has been transmitted, and

5   the target SCSI device has given up control of the SCSI bus, the target SCSI device then again arbitrates for control of the SCSI bus to initiate the status phase of the I/O transaction 714. In this phase, the SCSI bus transitions from the BUS FREE state through the ARBITRATION, RESELECTION, MESSAGE IN, STATUS, MESSAGE IN and BUS FREE states, as illustrated in FIG. 3C, in order to send a

10 SCSI status byte from the target SCSI device to the target SCSI-bus adapter. Upon receiving the status byte, the target SCSI-bus adapter packages the status byte into an FCP_RSP single-frame sequence 716 and transmits the FCP_RSP single-frame sequence back to the initiator through the FC. This completes the read I/O transaction.

15   In many computer systems, there may be additional internal computer buses, such as a PCI bus, between the target FC host adapter and the target SCSI-bus adapter. In other words, the FC host adapter and SCSI adapter may not be packaged together in a single target component. In the interest of simplicity, that additional interconnection is not shown in FIGS. 7A-B.

20   FIG. 7B shows, in similar fashion to FIG. 7A, a mapping between FCP sequences and SCSI bus phases and states during a write I/O transaction indicated by a FCP_CMND frame 718. FIG. 7B differs from FIG. 7A only in the fact that, during a write transaction, the FCP_DATA frames 722-725 are transmitted from the initiator to the target over the FC and the FCP_XFER_RDY

25 single-frame sequence 720 sent from the target to the initiator 720 is not optional, as in the case of the read I/O transaction, but is instead mandatory. As in Fig. 7A, the write I/O transaction includes when the target returns an FCP_RSP single-frame sequence 726 to the initiator.

Arbitrated Loop Initialization

As discussed above, the FC frame header contains fields that specify the source and destination fabric addresses of the FC frame. Both the D_ID and the S_ID are 3-byte quantities that specify a three-part fabric address for a particular FC port. These three parts include specification of an FC domain, an FC node address, and an FC port within the FC node. In an arbitrated loop topology, each of the 127 possible active nodes acquires, during loop initialization, an arbitrated loop physical address ("AL_PA"). The AL_PA is a 1-byte quantity that corresponds to the FC port specification within the D_ID and S_ID of the FC frame header. Because there are at most 127 active nodes interconnected by an arbitrated loop topology, the single byte AL_PA is sufficient to uniquely address each node within the arbitrated loop.

The loop initialization process may be undertaken by a node connected to an arbitrated loop topology for any of a variety of different reasons, including loop initialization following a power reset of the node, initialization upon start up of the first node of the arbitrated loop, subsequent inclusion of an FC node into an already operating arbitrated loop, and various error recovery operations. FC arbitrated loop initialization comprises seven distinct phases. FIG. 8 shows a diagram of the seven phases of FC arbitrated loop initialization. FIG. 9 shows the data payload of FC frames transmitted by FC nodes in an arbitrated loop topology during each of the seven phases of loop initialization shown in FIG. 9. The data payload for the FC frames used in each of the different phases of loop initialization comprises three different fields, shown as columns 902-904 in FIG. 9. The first field 902 within each of the different data payload structures is the LI_ID field. The LI_ID field contains an 16-bit code corresponding to one of the seven phases of group initialization. The LI_FL field 903 for each of the different data payload layouts shown in FIG. 9 contains various flags, including flags that specify whether the final two phases of loop initialization are supported by a particular FC port. The TL supports all seven phases of loop initialization. Finally, the data portion of

the data payload of each of the data payload layouts 904 contains data fields of varying lengths specific to each of the seven phases of loop initialization. In the following discussion, the seven phases of loop initialization will be described with references to both FIGS. 8 and 9.

5   In the first phase of loop initialization 802, called "LISM," a loop initialization master is selected. This first phase of loop initialization follows flooding of the loop with loop initialization primitives ("LIPs"). All active nodes transmit an LISM FC arbitrated loop initialization frame 906 that includes the transmitting node's 8-byte port name. Each FC port participating in loop

10   initialization continues to transmit LISM FC arbitrated loop initialization frames and continues to forward any received LISM FC arbitrated loop ·initialization frames to subsequent FC nodes in the arbitrated loop until either the FC port detects an FC frame transmitted by another FC port having a lower combined port address, where a combined port address comprises the D_ID, S_ID, and 8-byte

15   port name, in which case the other FC port will become the loop initialization master ("LIM"), or until the FC port receives back an FC arbitrated loop initialization frame that that FC port originally transmitted, in which case the FC port becomes the LIM. Thus, in general, the node having the lowest combined address that is participating in the FC arbitrated loop initialization process becomes

20   the LIM. By definition, an FL_PORT will have the lowest combined address and will become LIM. At each of the loop initialization phases, loop initialization may fail for a variety of different reasons, requiring the entire loop initialization process to be restarted.

            Once an LIM has been selected, loop initialization proceeds to the

25   LIFA phase 804, in which any node having a fabric assigned AL_PA can attempt to acquire that AL_PA. The LIM transmits an FC arbitrated loop initialization frame having a data payload formatted according to the data payload layout 908 in FIG. 9. The data field of this data layout contains a 16-byte AL_PA bit map. The LIM sets the bit within the bit map corresponding to its fabric assigned AL_PA, if

the LIM has a fabric assigned AL_PA. As this FC frame circulates through each FC port within the arbitrated loop, each FC node also sets a bit in the bit map to indicate that FC nodes fabric-assigned AL_PA, if that node has a fabric assigned AL_PA. If the data in the bit map has already been set by another FC node in the

5   arbitrated loop, then the FC node must attempt to acquire an AL_PA during one of three subsequent group initialization phases. The fabric assigned AL_PAs provide a means for AL_PAs to be specified by an FC node connected to the arbitrated loop via an FL_Port.

In the LIPA loop initialization phase 806, the LIM transmits an FC

10   frame containing a data payload formatted according to the data layout 910 in FIG. 9. The data field contains the AL_PA bit map returned to the LIM during the previous LIPA phase of loop initialization. During the LIPA phase 910, the LIM and other FC nodes in the arbitrated loop that have not yet acquired an AL_PA may attempt to set bits within the bit map corresponding to a previously acquired

15   AL_PA saved within the memory of the FC nodes. If an FC node receives the LIPA FC frame and detects that the bit within the bit map corresponding to that node's previously acquired AL_PA has not been set, the FC node can set that bit and thereby acquire that AL_PA.

The next two phases of loop initialization, LIHA 808 and LISA 810

20   are analogous to the above-discussed LIPA phase 806. Both the LIHA phase 808 and the LISA phase 810 employ FC frames with data payloads 912 and 914 similar to the data layout for the LIPA phase 910 and LIFA phase 908. The bit map from the previous phase is recirculated by the LIM in both the LIHA 808 and LISA 810 phases, so that any FC port in the arbitrated loop that has not yet acquired an

25   AL_PA may attempt to acquire either a hard assigned AL_PA contained in the port's memory, or, at last resort, may obtain an arbitrary, or soft, AL_PA not yet acquired by any of the other FC ports in the arbitrated loop topology. If an FC port is not able to acquire an AL_PA at the completion of the LISA phase 810, then that FC port may not participate in the arbitrated loop. The FC-AL-2 standard

contains various provisions to enable a nonparticipating node to attempt to join the arbitrated loop, including restarting the loop initialization process.

In the LIRP phase of loop initialization 812, the LIM transmits an FC frame containing a data payload having the data layout 916 in FIG. 9. The data field 917 of this data layout 916 contains a 128-byte AL_PA position map. The LIM places the LIM's acquired AL_PA, if the LIM has acquired an AL_PA, into the first AL_PA position within the AL_PA position map, following an AL_PA count byte at byte 0 in the data field 917, and each successive FC node that receives and retransmits the LIRP FC arbitrated loop initialization frame places that FC node's AL_PA in successive positions within the AL_PA position map. In the final loop initialization phase LILP 814, the AL_PA position map is recirculated by the LIM through each FC port in the arbitrated loop technology so that the FC ports can acquire, and save in memory, the completed AL_PA position map. This AL_PA position map allows each FC port within the arbitrated loop to determine its position relative to the other FC ports within the arbitrated loop.

The SCSI-3 Enclosure Services Commands

During the past decade, it has become increasingly popular for computer peripheral manufacturers to include a number of different peripheral devices within a single enclosure. One example of such enclosures is a redundant array of inexpensive disks ("RAID"). By grouping a number of different peripheral devices within a single enclosure, the peripheral manufacturer can achieve certain economies of manufacture. For example, all of the peripheral devices within the enclosure may share one or more common power supplies, cooling apparati, and interconnect media. Such enclosures may provide a collective set of resources greater than the resource represented by individual peripheral devices. In addition, individual peripheral devices may be swapped in and out of the enclosure while the other peripheral devices within the enclosure continue to operate, a process known as hot-swapping. Finally, banks of such enclosures may

be used for storage redundancy and mirroring in order to achieve economical, highly available resources.

Fig. 10 illustrates a simple multi-peripheral devices enclosure. The enclosure 1002 includes a power supply 1004, a cooling fan 1006, four disk drives 1008-1011. A circuit board 1014 within the enclosure includes a processor 1016, an internal bus 1018, and an interconnection medium 1020 that interconnects the processor 1016, the disk drive 1008-1011, and a port 1022 through which the enclosure 1002 can be connected to a host computer (not shown). The host computer may, in some systems, individually address and interact with the disk drives 1008-1011 as well as with the processor 1016, or may instead interact with the enclosure 1002 as if the enclosure represented one very large disk drive with a single address base. The processor 1016 generally runs a process that may monitor status of each of the peripheral devices 1008-1011 within the enclosure 1002 as well as the status of the power supply 1004 and the cooling fan 1006. The processor 1016 communicates with the power supply 1004 and the cooling fan 1006 via an internal communications medium such as, in Fig. 10, an internal bus 1018.

In order to facilitate host computer access to information provided by various components within an enclosure, such as the power supply 1004 and the cooling fan 1006 and in order to provide the host computer with the ability to individually control various components within the enclosure, a SCSI command set has been defined as a communications standard for communications between a host computer and an enclosure services process running within an enclosure, such as enclosure 1002 in Fig. 10. The SCSI Enclosure Services ("SES") command set is described in the American National Standard for Information Technology Standards Document NCITS 305-199X. The SES command set will be defined in a reference standard that is currently still under development by the X3T10 Committee.

Fig. 11 illustrates the basic communications paradigm represented by the SES command set. A host computer 1102 sends an SES command 1104 to an

enclosure services process 1106 running within an enclosure 1108. In Fig. 10, for example, the enclosure services process runs on processor 1016. The enclosure services process 1106 interacts with various components 1110-1113 within the enclosure 1108 and then returns a response 1114 to the SES command sent to the enclosure services process 1106 by the host computer 1102.

5

There are a number of different types of SES commands and responses to SES commands. The above cited ANSI standard documents may be consulted for details on the various types of commands and responses. In general, the bulk of communications traffic between a host computer 1102 and an enclosure services process 1106 involves two basic commands: (1) the SEND DIAGNOSTICS command by which the host computer transmits control information to the enclosure services process; and (2) the RECEIVE DIAGNOSTIC RESULTS command by which the host computer solicits from the enclosure services process information, including state information, about the various components within an enclosure.

10

15

The host computer transmits a SEND DIAGNOSTICS command to the enclosure services process via an enclosure control page. The layout for an enclosure control page is shown below in Table 1.

Table 1

Enclosure control page

Bits / Bytes	7	6	5	4	3	2	1	0
0	PAGE CODE (02H)							
1	Reserved				INFO	NON-CRIT	CRIT	UN-RECOV
2	(MSB) PAGE LENGTH (N-3)							
3	(LSB)							
4-7	GENERATION CODE							
8-11	OVERALL CONTROL (first element type)							
12-15	ELEMENT CONTROL (first element of first element type)							
	***							
(4 bytes)	ELEMENT CONTROL (last element of first element type)							
(4 bytes)	OVERALL CONTROL (second element type)							
(4 bytes)	ELEMENT CONTROL (first element of second element type)							
	***							
n-3 to n	ELEMENT CONTROL (last element of last element type)							

The enclosure control page includes an OVERALL CONTROL field for each type of component within an enclosure and an ELEMENT CONTROL field for each discrete component within an enclosure. ELEMENT CONTROL fields for all components of a particular type are grouped together following the OVERALL CONTROL field for that type of component. These control fields have various formats depending on the type of component, or element. The formats for

the control fields of the enclosure control page will be described below for several types of devices. The types of elements currently supported by the SES command set are shown below in Table 2

Table 2

Type Code	Type of element	Type Code	Type of element
00h	Unspecified	0Dh	Kay pad entry device
01h	Device	0Eh	Reserved
02h	Power supply	0Fh	SCSI port/transceiver
03h	Cooling element	10h	Language
04h	Temperature sensors	11h	Communication port
05h	Door lock	12h	Voltage sensor
06h	Audible alarm	13h	Current sensor
07h	Enclosure services controller electronics	14h	SCSI target port
08h	SCC controller electronics	15h	SCSI initiator port
09h	Nonvolatile cache	16h	Simple sub-enclosure
0Ah	Reserved	17-7Fh	Reserved
0Bh	Uninterruptible power supply	80h-FFh	Vendor-specific codes
0Ch	Display	***	

When a host computer issues a RECEIVED DIAGNOSTIC RESULTS command to the enclosure services process, the enclosure services process collects status information from each of the components, or elements, within the enclosure and returns an enclosure status page to the host computer that

contains the collected status information. The layout of the enclosure status page is shown below in Table 3.

Table 3

Enclosure status page

Bits / Bytes	7	6	5	4	3	2	1	0
0	PAGE CODE (02H)							
1	Reserved			INVOP	INFO	NON-CRIT	CRIT	UNREC OV
2	(MSB)							
3	PAGE LENGTH (n-3) (LSB)							
4-7	(MSB) GENERATION CODE (LSB)							
8-11	OVERALL STATUS (first element type)							
12-15	ELEMENT STATUS (first element of first element type)							
	***							
(4 bytes)	ELEMENT STATUS (last element of first element type)							
(4 bytes)	OVERALL STATUS (second element type)							
(4 bytes)	ELEMENT STATUS (first element of second element type)							
	***							
n-3 to n	ELEMENT STATUS (last element of last element type)							

As with the enclosure control page, described above, the enclosure status page contains fields for particular components, or elements, grouped together following an overall field for that type of component. Thus, the enclosure status page contains an OVERALL STATUS field for each type of element followed by

5   individual ELEMENT STATUS fields for each element of a particular type within the enclosure. The status fields vary in format depending on the type of element. The status field formats for several devices will be illustrated below.

The host computer can issue a RECEIVED DIAGNOSTICS RESULTS command with a special page code in order to solicit from the enclosure

10   services process a configuration page that describes the enclosure and all the components, or elements, within the enclosure. Table 4, below, shows the layout of a configuration page.

Table 4

Configuration page

Component name	Bytes	Field Name
Diagnostic page header		
Generation code		
Enclosure descriptor header	8	Reserved
	9	SUB-ENCLOSURE IDENTIFIER
	10	NUMBER OF ELEMENT TYPES SUPPORTED (T)
	11	ENCLOSURE DESCRIPTOR LENGTH (m)

Enclosure descriptor	12-19	ENCLOSURE LOGICAL IDENTIFIER
	2-27	ENCLOSURE VENDOR IDENTIFICATION
	28-43	PRODUCT IDENTIFICATION
	44-47	PRODUCT REVISION LEVEL
	48 - (11+m)	VENDOR-SPECIFIC ENCLOSURE INFORMATION
Type descriptor header list	(4 bytes)	TYPE DESCRIPTOR HEADER (first element type)
		***
	(4 bytes)	TYPE DESCRIPTOR HEADER (T^{Th} element type)
Type descriptor text	variable	TYPE DESCRIPTOR TEXT (first element type)
		***
	last byte=n	TYPE DESCRIPTOR TEXT (T^{Th} element type)

The configuration page includes an enclosure descriptor header and an enclosure descriptor that describes the enclosure, as a whole, as well as a type descriptor header list that includes information about each type of component, or element, included in the enclosure and, finally, a type descriptor text list that contains descriptor text corresponding to each of the element types.

Tables 5A-B, below, show the format for an ELEMENT control field in the enclosure control page for a cooling element, such as a fan.

Table 5A

Cooling element for enclosure control pages

Bits / Bytes	7	6	5	4	3	2	1	0
0	COMMON CONTROL							
1-2	Reserved							
3	Rsrvd	RQST FAIL	RQST ON	Reserved		REQUESTED SPEED CODE		

Table 5B

REQUESTED SPEED CODE values

Speed Code	Description
000b	Reserved
001b	Fan at lowest speed
010b	Fan at second lowest speed
011b	Fan at speed 3
100b	Fan at speed 4
101b	Fan at speed 5
110b	Fan at intermediate speed
111b	Fan at highest speed

Bit fields within the ELEMENT control field allow the host computer to specify to the enclosure services process certain actions related to a particular cooling element. For example, by setting the RQST FAIL bit, the host computer specifies that a visual indicator be turned on to indicate failure of the cooling element. By setting the RQST ON field, host computer requests that the

cooling element be turned on and remain on. The REQUESTED SPEED CODE field allows the host computer to specify a particular cooling fan speed at which the cooling element should operate. Table 5B includes the different fan speed settings that can be specified in the requested speed code field.

5   Tables 6A-B, below, show the layout for a cooling ELEMENT STATUS field within an enclosure status page, shown above in Table 3.

Table 6A

Cooling element for enclosure status pages

Bits Bytes	7	6	5	4	3	2	1	0
0	COMMON STATUS							
1-2	Reserved							
3	Rsrvd	FAIL	RQSTE D ON	OFF	Rsrvd	ACTUAL SPEED CODE		

10   **Table 6B**

ACTUAL SPEED CODE values

Speed Code	Description
000b	Fan stopped
001b	Fan at lowest speed
010b	Fan a second lowest speed
011b	Fan at speed 3
100b	Fan at speed 4
101b	Fan at speed 5
110b	Fan at intermediate speed
111b	Fan at highest speed

The various bit fields within the cooling ELEMENT STATUS field, shown in Table 6A, indicate to the host computer the state of the particular cooling element, or fan. When the FAIL bit is set, the enclosure services process is indicating that the failure indication for a particular fan has been set on. When the RQSTED ON bit is set, the enclosure services process indicates to the host computer that the fan has been manually turned on or has been requested to be turned on via a SEND DIAGNOSTICS command. When the OFF bit is set, the enclosure services process indicates to the host computer that the fan is not operating. The enclosure services process may indicate to the host computer, via the ACTUAL SPEED CODE field, the actual speed of operation of the fan. Actual speed code values are shown above in Table 6B.

A layout for the ELEMENT CONTROL field for a power supply within the enclosure control page, shown above in Table 1, is shown below in Table 7A. An ELEMENT STATUS field for a power supply element that is included in an enclosure status page, shown above in Table 3, is shown below in Table 7B.

Table 7A

Power supply element for enclosure control page

Bits Bytes	7	6	5	4	3	2	1	0
0	COMMON CONTROL							
1-2	Reserved							
3	Rsrvd	RQST FAIL	RQST ON	Reserved				

Table 7B

Power supply element for enclosure status pages

Bits / Bytes	7	6	5	4	3	2	1	0
0	COMMON STATUS							
1	Reserved							
2	Reserved				DC over-voltage	DC under-voltage	DC over-current	Rsrvd
3	Rsrvd	FAIL	RQSTED ON	OFF	OVRTM P FAIL	TEMP WARN	AC FAIL	DC FAIL

5   Many of the fields in the power supply control and status fields are similar to those in the cooling element control and status fields of Tables 5A and 6A, and will not be further discussed. The power supply status field also includes bit fields to indicate under-voltage, over-voltage, over-current, power failure, and other temperature conditions.

10   The SES command set and SES protocol specify a standard SCSI communication between a host computer and an enclosure including multiple peripheral devices. The SES protocol allows the host computer to control operation of individual peripheral devices within the enclosure and also to acquire information about the status of operation of the peripheral devices

15

Multi-Disk Enclosures

    The highbandwidth and flexible connectivity provided by the FC, along with the ability of the FC to support the SES command set and protocol, have made the FC an attractive communications medium for interconnecting host

processors with enclosures containing multiple peripheral devices and for interconnecting the multiple peripheral devices within enclosures. In the following discussions, enclosures will be described and represented as containing multiple disk drives. However, the described techniques and approaches for interconnecting

5   multiple disk drives within an enclosure, and for interconnecting enclosures and host computers, are equally applicable for other types of peripheral devices.

Fig. 12 is a simplified illustration of the design used by manufacturers of certain currently-available FC-based multi-disk enclosures. The enclosure 1202 is shown in Fig. 12 containing 8 disks drives 1204-1211. The disk

10   drives are plugged into, and interconnected by, a backplane 1212. A multi-component circuit board 1214 is also plugged into the backplane 1212. Two giga-bit interface converters ("GBICs") 1216 and 1218 provide external fibre optic cable connection to the enclosure 1202. The circuit board 1214 contains a processor 1220 and a number of port bypass circuits ("PBCs") 1222-1229 that are

15   interconnected by an internal FC loop 1230. An enclosure services process runs on the processor 1220 to allow the host computer (not shown) to control various additional components within the enclosure, such as fans, power supplies, temperature sensors, etc., as discussed in the previous subsection. The individual disk drives 1204-1211 of the enclosure may be replaced, removed, or added during

20   operation of the other disk drives of the enclosure. Hot-swapping is made possible in the currently-available systems illustrated in Fig. 12, by the port bypass circuits 1222-1229. When a disk is present and functioning, the FC signal passes from the FC loop 1230 through the port bypass circuit (for example, port bypass circuit 1225) to the disk drive (for example, disk drive 1207). When a disk drive is

25   removed, the port bypass circuit instead routes the FC signal directly to the next port bypass circuit or other component along the FC loop 1230. For example, if disk drive 1207 is removed by hot-swapping, FC signals will pass from disk drive 1206 through port bypass circuit 1224 to port bypass circuit 1225 and from port bypass circuit 1225 directly to port bypass circuit 1226.

A single GBIC (for example, GBIC 1216) allows connection of the enclosure to a host computer via an optical fibre. A second GBIC (for example, GBIC 1218) may allow an enclosure to be daisy-chained to another enclosure, thereby adding another group of disk drives to the fibre channel loop 1230. When a second GBIC is present, and no further enclosures are to be daisy-chained through the second GBIC, a loop-back connector, or terminator, is normally plugged into the second GBIC to cause FC signals to loop back through the enclosure and, ultimately, back to the host computer.

Fig. 13A is a schematic representation of a port bypass circuit, such as port bypass circuits 1222-1229 in Fig. 12. An input FC signal ("IN") 1302 passes through a summing amplifier 1304 to convert the differentially-encoded FC signal into a linear signal used within the PBC circuitry. Summing amplifiers 1306-1308 are similarly employed to interconvert linear and differential signals. The converted input signal 1310 is split and passed to a buffered output ("Pout") 1312 and to a multiplexer 1314. A second FC input signal ("Pin") 1316 passes through summing amplifier 1307 and is input to the multiplexer 1314. The FC output signal ("OUT") 1318 from the multiplexer 1314 is controlled by the SEL control input line 1320. When the SEL control input line is asserted, the multiplexer 1314 passes the Pin input 1316 to the output signal 1318. When the SEL control input line is de-asserted, the multiplexer 1314 passes the IN input signal 1302 to the output signal OUT 1318.

Fig. 13B illustrates the connection of a disk drive to a fibre channel loop via a port bypass circuit. In the interest of brevity, the components of the port bypass circuit in 13B that are the same as components shown in Fig. 13A will be labeled in 13B with the same labels used in Fig. 13A, and descriptions of these components will not be repeated. The disk drive 1322 receives an input signal IN 1302 from the fibre channel loop via the Port signal 1312. When the disk drive asserts the SEL control signal 1320, the disk drive provides the signal Pin 1316 that is passed by the multiplexer 1314 to the output signal OUT 1318 that is transmitted

via the FC loop to the next FC port in the direction of the FC signal. When the SEL control signal 1320 is de-asserted, the disk drive 1322 is bypassed, and the input signal IN 1302 is passed as the output signal OUT 1318 to the next FC port in the direction of the FC signal. The disk drive 1322 asserts the SEL control signal

5   when it is securely mounted in the enclosure, connected to the backplane, and functionally ready to inter-operate with the FC loop. When the disk drive 1322 is absent, or not functionally ready to inter-operate with the FC loop, the SEL control line 1320 is de-asserted and the FC signal bypasses the disk drive. When the disk drive is hot-swapped into or out of an on-line enclosure, the FC loop that

10   interconnects the functioning disk drives must undergo re-initialization, as discussed above, but the ensuing interruption is relatively slight, and any interrupted data transfers are recovered. However, there are different possible failure modes of disk drives that can degrade or disable operation of the FC loop and that cannot be detected and bypassed by the essentially passive PBC. For

15   example, a disk drive may intermittently transmit spurious signals, or may fail to yield control of the FC loop after transmitting requested data. Thus, although the passive PBCs allow for hot-swapping of disk drives, they do not provide the high level of component malfunction detection and recovery necessary in high-availability systems.

20   **The Multi-Disk Enclosure of the Present Invention**

The method and system of the present invention are related to a new type of multi-peripheral-device enclosure that provides increased reliability, increased fault tolerance, and higher availability. Again, as in the previous subsection, this new multi-peripheral-device enclosure will be illustrated and

25   described in terms of a multi-disk enclosure. However, the techniques and methods of the present invention apply generally to enclosures that may contain different types of peripheral devices in different combinations. The method and system of the present invention will be discussed with regard to enclosures based on FC interconnection between the host computer and the enclosure as well as between

various peripheral devices within the enclosure. However, other types of communications media may be employed in place of the FC. Finally, the method and system of the present invention are discussed with regard to a multi-disk enclosure in which the SES command set and protocol provide component-level control to the host computer. However, this component-level control may be provided by other types of protocols and command sets.

Fig. 14 shows a highly available enclosure that incorporates techniques related to the present invention. The highly available enclosure ("HAE") shown in Fig. 14 includes 8 disk drives 1402-1409. The disk drives 1402-1409 are plugged into a backplane 1412 that interconnects the disk drives with other components in the HAE, and that also interconnects certain of the other components in the HAE independently from the disk drives. The backplane 1412 is passive. It contains no active components, such as processing elements, and is thus highly unlikely to become a point of failure within the HAE. The two link control cards ("LCCs") 1414 and 1416 are coupled to the backplane. The two LCCs are essentially identical. Only the components included in the top LCC 1414 will be described and labeled. An LCC contains two GBICs 1418 and 1420, a number of port bypass circuits 1422-1424, and several port bypass circuit chips 1426 and 1428, each of which contains four separate port bypass circuits. The port bypass circuits and port bypass circuit chips are interconnected both by an FC loop, indicated in Fig. 14 by the single heavy line, for example line 1430 interconnecting port bypass circuits 1422 and 1423, and a port bypass circuit bus 1432. In Fig. 14, port bypass circuits may be shown interconnected by both a port bypass circuit bus as well as an FC loop as, for example, the interconnection between port bypass circuits 1422 and 1423. The port bypass circuit chips 1426 and 1428 fan out Pout, Pin, and SEL control line signals, represented collectively in Fig. 14 by a single line, such as line 1434, to the 8 disk drives 1402-1409. Each port bypass circuit chip controls FC loop access to four disk drives. The LCC contains a processor 1436, which runs an enclosure services process and other

control programs. This processor 1436 includes circuitry that implements an FC port as well as ports to three different internal busses. One of the internal busses 1438, in a preferred embodiment an I²C bus, interconnects the processor 1436 with PBC controller chips 1440 and 1442 and with other

5   components such as temperature sensing devices and power monitoring devices 1444 and 1446. The processor on one LCC 1436 is interconnected with the processor on the other LCC 1448 by two separate internal busses 1450 and 1452 that run through the backplane 1412.

       The HAE is highly redundant. The disk drives 1402-1409 are

10   interconnected by two separate FC loops implemented, in part, on the two LCC cards 1414 and 1416. Thus, if one FC loop fails, a host computer (not shown) can nonetheless access and exchange data with the disk drives in the enclosure via the other FC loop. In similar fashion, if one internal bus that interconnects the two processors 1436 and 1448 fails, the two processors can communicate via the other

15   internal bus. Although not shown in Fig. 14, the HAE includes dual power supplies and other redundant components. Each of the two processors controls one of the two redundant components, such as one power supply, to ensure that a failing processor is not able to shut down both power supplies and thus disable the HAE. The port bypass circuits, as in the currently-available enclosures described

20   above, allow for hot-swapping of disk drives. However, because the port bypass circuits are themselves controlled by port bypass circuit controllers 1440 and 1442, additional higher-level control of the components can be achieved. For example, a faulty disk drive can be identified and isolated by a software routine running on the processor 1436 which can then signal a port bypass circuit controller to forcibly

25   bypass a particular disk drive. Redundant environmental monitors allow for vigilant fault-tolerant monitoring of the conditions within the HAE of both processors. Failure of any particular sensor or interconnecting internal bus will not produce a failure of the entire HAE..

Fig. 15A illustrates control of a port bypass circuit by a port bypass circuit control chip. The circuit illustrated in Fig. 15A is similar to the circuit shown in Fig. 13B above. However, the control signal line, in this circuit designated the "SD" control signal line 1502, does not directly control output of the

5   multiplexer 1504. Instead, the SD control signal line 1502 is input to a PBC control circuit 1506. This PBC control circuit may be implemented by a microprocessor or may be implemented based on state-machine logic. The PBC control circuit 1506 outputs a forced bypass control circuit line ("FB") that determines, as in the circuit in 13B, whether the input signal IN 1508 is passed

10 through to the output signal OUT 1510 or whether, instead, the Pin signal 1512 is passed by the multiplexer 1504 to the output signal OUT 1510. The PBC control circuit 1506 can also exchange data with the microprocessor 1508 via a serial bus 1510 or some other type of communication media. The microprocessor 1508 can indicate to the PBC control circuit 1506 that the PBC control circuit 1506

15 should assert the FC control signal 1503 in order to bypass the disk drive 1514. Thus, in the circuit shown in Fig. 15A, several additional levels of control are available besides the control exerted by the disk 1514 via signal line SD 1502. The PBC control circuit 1506 may forcibly bypass the disk 1514 according to an internal set of rules, and a program running within the microprocessor 1508 can

20 cause the disk 1514 to be forcibly bypassed via data transmitted to the PCB control circuit 1506. These additional levels of control allow for microprocessor-controlled bypass of individual disk drives following detection of disk malfunction or critical events signaled by environmental monitors and other such sensors.

Fig. 15B shows an example of the PBC control circuit implemented

25 in hardware. A D flip-flop 1516 outputs the forced bypass signal FB 1518. The D flip-flop changes state upon receiving a strobe input signal 1520. The D flip-flop receives input from the SD control signal line 1522 and the write_data 1524 input from the microprocessor. The strobe signal is generated whenever the SD control line changes state or whenever there is a microprocessor write operation. The D

flip-flop can be set or cleared based on changes either to the SD input 1512, or by changes to write data 1524 input from a microprocessor. The forced bypass signal FB tracks the SD control signal 1522, but may be overridden by microprocessor control. Thus, the control circuit of 15B, when included as PBC control

5   circuit 1506 in Fig. 15A, allows circuit 15A to function identically to the circuit of Fig. 13A except in the case that the microprocessor elects to forcibly bypass the disk, rather than depend on the disk to bypass itself.

The enhanced PBC control circuit of Fig. 15A is also used in the HAE to implement various shunting operations. For example, PBC 1422 and 1423

10 in Fig. 14 can be controlled by PBC circuit controllers 1440 and 1442 to bypass GBICs 1418 and 1420, respectively. Figs. 16A-B illustrate the usefulness of implementing a shunting operation in order to bypass a GBIC. In Fig. 16A, two HAEs 1602 and 1604, are schematically shown daisy-chained together via a single FC loop 1606. The FC optical fibre incoming from the host computer (not shown)

15 connects through a first GBIC 1608 to the first HAE 1602. The FC loop exits the first HAE 1602 at GBIC 1610 and enters the second HAE 1604 at GBIC 1612. Finally, the FC loop exits the second HAE 1604 at GBIC 1614 and returns to the host computer via a return path. The FC circuit is looped back from GBIC 1614 using an external loop back hood 1616.

20 There are problems associated with the simple form of daisy-chaining illustrated in Fig. 16A. First, certain malfunctions within the second HAE 1604 might bring down the entire FC loop, including the first HAE 1602. Thus, HAEs cannot be readily isolated and bypassed when they are daisy-chained according to the scheme of Fig. 16A. Also, the external loop back hood 1616 is an

25 additional component that adds cost to the overall system, may cause problems in installation, and provides yet another source of single-point failure.

The above-noted deficiencies related to the daisy-chaining of Fig. 16A can be overcome using shunt operations controlled by PBC control logic circuits. Fig. 16B shows a HAE, schematically diagramed as in Fig. 16A, with the

functionality provided by the external loop back hood 1616 of Fig. 16A instead implemented via a PBC. In Fig. 16B, the rightmost GBIC 1618 of HAE 1620 is controlled by PBC 1622. PBC 1622 is, in turn, controlled by a PBC controller 1624 which may, in turn, be controlled by the microprocessor (not

5   shown). The return FC signal 1626 is fed back into the PBC controller 1624, following conversion, as a control signal line 1628. When the GBIC 1618 is connected to a fibre optic cable that is, in turn, connected to another HAE, the FC return signal 1626 causes the control signal line 1628 to be asserted, and causes the PBC controller 1624 to control the PBC 1622 to pass FC signals between the HAE

10   and an external additional HAE. However, when the HAE is not connected via GBIC 1618 and a fibre optic cable to another HAE, the control signal line 1628 will be de-asserted, causing the PBC controller 1624 to control the PBC 1622 to bypass the GBIC 1618 and thus looping the FC signal back via a return path to the host computer. This mechanism eliminates the need for an external loop back

15   hood 1616, and provides for automatic sensing of daisy-chained enclosures. Moreover, if an enclosure downstream from HAE 1620 malfunctions, the host computer (not shown) may interact with the microprocessor within the HAE (also not shown) to direct the PBC controller 1624 to forcibly bypass the GBIC 1618 via the PBC 1622, thus removing downstream enclosures from the FC loop. Thus,

20   defective enclosures can be isolated and removed via microprocessor-controlled shunting of GBICs.

The Present Invention

The present invention relates to the detection and isolation of malfunctioning disk drives within HAEs as well as the detection and isolation of

25   malfunctioning HAEs within a daisy-chain of HAEs connected via FC arbitrated loops to a host computer. The basic strategy represented by the present invention is for each HAE, upon power up, to conduct a self-test prior to joining the FC arbitrated loops that interconnect a series of HAEs with a host computer. In this fashion, each HAE guarantees a certain level of functionality and reliability prior to

joining the FC arbitrated loops. If a particular disk drive within the HAE is malfunctioning, that disk drive can be bypassed via a PBC circuit controller so that the malfunctioning disk drive does' not end up being joined to the FC arbitrated loops. If an entire HAE malfunctions, either that HAE, or other downstream

5   HAEs, can be bypassed to allow upstream HAEs to join the FC arbitrated loops. Alternatively, if the signal direction of the two FC arbitrated loops that interconnect a series of HAEs with the host computer have opposite directions, then a single malfunctioning HAE can be isolated and removed while still allowing host computer access to the remaining HAEs. HAEs downstream from a

10   malfunctioning HAE and removed in one FC arbitrated loop are, in the other FC arbitrated loop, upstream from the malfunctioning HAE and are thus accessible to the host computer via the second FC arbitrated loop.

The HAE arbitrated loop self-test management process will be described below in C++-like pseudocode. A number of classes will be presented

15   that represent different functionalities available to a self-test management process ("STMP") that runs on one or both processors within each HAE. The methods for these classes are declared, but no implementations are given. Implementation of these methods may involve purely hardware or hardware/software combinations that are straightforwardly implemented in view of the above detailed description of

20   the HAE architecture. Following these class declarations, an implementation of a power up routine is provided to illustrate and describe a preferred embodiment of the present invention. Finally, an interrupt service routine is provided to show how run-time malfunctions of disk drives within the HAE are detected and isolated in order to achieve continued operation necessary for highly available systems.

25   It should be noted that the pseudo-code implementations, provided below, can be written in any number of different languages and in an almost limitless number of ways. Implementations depend on particular choices made for hardware components and hardware configuration of the HAEs as well as choices on how the HAE and the entire host computer/HAEs systems should respond to

various malfunctions and error conditions. The pseudo-code routines, provided below, show one embodiment that illustrates the use of unique hardware features of the HAE to implement a STMP that runs on one or both processors within the HAE and that provides increased reliability to the system as a whole.

5   The STMP, running on a microprocessor within the HAE, may employ three main sets of functionalities: (1) directives to PBC circuit controllers; (2) FC arbitrated loop operations; and (3) SES protocol commands. All three types of functionalities have been introduces and describes in previous subsections. These three sets of functionalities are encapsulated in three class declarations that

10   follow, along with a few additional functionalities, enumerations, and constants.

```
1    class set
2    {
3      Boolean in(int e);
4    }
5
6    class PBCcontroller
7    {
8      void bypassPrimaryPort();
9      void bypassExpansionPort();
10     void bypassDisk(int diskno);
11     void unBypassPrimaryPort();
12     void unBypassExpansionPort();
13     void unBypassDisk (int diskno);
14     Boolean diskInstalled (int diskno);
15     Boolean signalDetectPrimary();
16     Boolean signalDetectExpansion();
17   }
18
19   class FC
20   {
21     void initializeLoop();
22     void send_LIPF7F7();
23     FCStatus receiveLIPStatus();
24     void report (int *buff);
25     int getUpstreamAlpa (int alpa);
26     int getDiskNo (int alpa);
27   }
28
```

```
29    class SES
30    {
32      void issueTestUnitReady(int diskno);
33      void issueInquiry(in diskno);
34      void issueReadCapacity (int diskno);
35      SESstatusreceiveSESStatus (int * buff);
36    }
37
38    enum FCstatus { LIP_RECEIVED, TIMED_OUT}
39    enum SESstatus { GOOD, BAD}
40    enum ERRORS { ENCLOSURE_FAULT, DISK_FAILURE,
41              TEST_UNIT_READY_FAILURE,
42              INQUIRY_FAILURE, READ_CAPACITY_FAILURE}
43
44    const int NUM_DISKS;
45    const int BUFFSIZE;
46
47    error (ERRORS e) {};
48
49    set OurAlpas;
```

The class "set," declared above on lines 1-4, is a generalized implementatation of an integer set. The method "in," declared above on line 3, returns a Boolean value indicating whether the integer supplied as argument "e" is contained within an instantiation of the set class.

The class "PBCcontroller," declared above on lines 6-17, encapsulates the PBC circuit controller functionality available to the STMP and provided by one of the two PBC circuit controllers on an LCC, such as PBC circuit controllers 1440 and 1442 in Fig. 14, for isolating disks and GBICs from an FC arbitrated loop and for including disks and GBICs into an FC arbitrated loop. In addition, the PBC circuit controller functionality allows the STMP to detect whether a disk is installed and to detect external FC signals received at a GBIC. In the pseudo-code implementation, the two GBICs on an LCC, for example GBICs 1418 and 1420 on LCC 1414 in Fig. 14, are separately classified. One GBIC, called the primary GBIC, represents the upstream GBIC most directly associated with the host computer, such as GBIC 1608 in Fig. 16A. The other GBIC, called the expansion GBIC, represents the downstream GBIC most closely

associated with a subsequent downstream HAE, such as GBIC 1610 in Fig. 16A. The GBIC along with FC protocol software running on a processor together constitutes a port. Thus, each LCC includes a primary port and an expansion port.

The method "bypassPrimaryPort," declared above on line 8, directs a PBC circuit controller to bypass the primary port comprising the primary GBIC and FC protocol logic running on the microprocessor of an LCC. Similarly, the method "bypassExpansionPort," declared above on line 9, allows the STMP to direct a PBC circuit controller to bypass the expansion port. The method "bypassDisk," declared above on line 10, allows the STMP to direct the PBC circuit controller to bypass the disk indicated by the integer argument "diskno." The method "unBypassPrimaryPort," declared above on line 11, allows the STMP to direct a PBC circuit controller to configure the primary port into an FC arbitrated loop. Similarly, the methods "unBypassExpansionPort" and "unBypassDisk," declared above on lines 12 and 13, allow the STMP to direct a PBC circuit controller to configure the expansion port and an indicated disk into the FC arbitrated loop. The method "diskInstalled," declared above on line 14, returns a Boolean value indicating whether the disk indicated by the integer argument "diskno" is installed within the HAE. The method "signalDetectPrimary," declared above on line 15, allows the STMP to direct a PBC circuit controller to determine whether a signal has been detected at the primary port, and the method "signalDetectExpansion," declared above on line 16, similarly allows the STMP to determine whether a signal has been detected by the expansion port. In the pseudo-code implementation, an instance of the PBC controller class represents the functionality available to the STMP from all PBC circuit controllers rather than on an LCC. For fault-tolerant operation, as described above, an LCC generally contains redundant PBC circuit controllers. Normally, the STMP may use one PBC circuit controller, under normal conditions, and fail over to a redundant PBC circuit controller when the first PBC circuit controller fails. Thus, it can be assumed that an instance of a PBC controller class implements PBC circuit

controller fail over and the proper direction of commands to the one of the two PBC circuit controllers currently being used to control PBC circuits.

The class "FC," declared above on lines 19-27, represents FC operations available to the STMP. The method "initializeLoop," declared above on

5   line 22, allows the STMP to initialize an internal arbitrated loop implemented on an LCC according to the FC loop initialization protocols described above in a previous subsection. The method "send_LIPF7F7," declared above on line 22, provides the STMP with the ability to send a particular type of loop primitive onto the internal FC arbitrated loop. Loop primitives comprise a number of bytes that indicate an

10   overall type of loop primitive and subclasses of loop primitives. The member function "send_LIPF7F7," sends a loop initialization primitive with second and third bytes containing the hexidecimal values "F7." This loop initialization primitive is normally used by an originating L_Port to acquire an AL_PA. A second type of loop initialization primitive contains the hexidecimal value "F8" in

15   the second byte and the hexidecimal value "F7" in the third byte, and is used by an L_port to indicate that a loop failure has been detected at its receiver.

The member function "receiveLIPStatus," declared above on line 23, allows the STMP to wait for reception of a loop primitive from the FC arbitrated loop. When the loop primitive is received, the member function

20   "receiveLIPStatus" returns a status of LIP_RECEIVED. If the member function "receiveLIPStatus" times out without receiving a loop initialization primitive, then the member function "receiveLIPStatus" returns a status of TIMED_OUT. The member function "report," declared above on line 24, allows the STMP to report to the host computer or internally store information contained within a buffer

25   pointed to by the integer pointer argument "buff." This member function allows information obtained by the STMP concerning the HAE and disk drives within the HAE to be reported back to the host computer or internally stored within the HAE. The member function "getUpstreamAlpa," declared above on line 25, returns the AL_PA of the L_Port directly upstream from the L_Port having the AL_PA

furnished by argument "alpa." This information is available from the loop map obtained by the STMP as a result of the final phase of FC arbitrated loop initialization described above in a previous section. Finally, the member function "getDiskNo," declared above on line 26, returns the disk number within a HAE

5   that corresponds to the AL_PA furnished in the argument "alpa." This information is available in stored information, including the loop map and an index of disk drives and corresponding AL_PAs.

The class "SES," declared above on lines29-35, represents the higher-level SES command set and protocol available to the STMP for controlling

10   and obtaining information from individual disk drives and other HAE components. For the current pseudo-code, only a few member functions are provided. A complete SES class would, in addition, include the commands and SES status page protocol described above in a previous subsection. The member "issueTestUnitReady," declared above on line 32, issues a TEST_UNIT_READY

15   command to the disk drive indicated by the integer argument "diskno." The member function "issueInquiry," declared above on line 33, issues an INQUIRY command to the disk drive indicated by the integer argument "diskno." The member function "issueReadCapacity," declared above on line 34, allows the STMP to issue a READ_CAPACITY command to the disk drive indicated by the

20   integer argument "diskno."

The SES command TEST_UNIT_READY solicits from an element, or component, a status indication indicating whether the element, or component, is online and ready for operation. The SES command INQUIRY solicits certain information about a particular element, or component, that is returned by the

25   element, or component, in a multi-byte buffer. The SES command READ_CAPACITY solicits from a particular element, or component, information about the storage capacity or other such capacity available from the component, returned in a multi-byte buffer.

The member function "receiveSESStatus," declared above on line 35, allows the STMP to wait for the return of information solicited by a TEST_UNIT_READY, INQUIRY, or READ_CAPACITY command to the receive that information in the buffer specified by the argument "buff." If no response is 5 forthcoming, receiveSESStatus times out and returns a status of BAD.

The different values for FC status and SES status are provided by the enumerations "FCstatus" and "SESstatus," declared above on lines 38 and 39. The enumeration "ERRORS," declared above on lines 40-42, includes different error conditions that can arise during execution of the STMP power up routine, to be 10 described below. The constants NUM_DISKS and BUFFSIZE, declared above on lines 44 and 45, represent the number of disks within a HAE and the size of the buffer required for SES information solicited by the various SES commands described above with relation to class "SES."

The function "error," declared above on line 47, represents a 15 generalized error handling routine that appropriately responds to the error indicated by argument "e." Many different types of error handling strategies can be employed for any particular type of error that arises. The STMP may report some errors to the host computer, attempt to isolate and recover from certain other errors within the HAE, or embark on more complex procedures to reinitialize the HAE or 20 conduct other recovery schemes. Finally, the set "OurAlpas," declared above on line 49, is a set of the AL_PAs corresponding to the disk drives within a HAE.

The STMP executes the following routine "powerUp" following initial power up of a HAE. This power-up routine, along with an interrupt service routine that follows the power-up routine, together comprise an embodiment of the 25 present invention.

```
1    void powerUp (PBC controller & pbc, FC & fc, SES & ses);
2    {
3
4        int buffer[BUFFSIZE];
5        pbc.bypassPrimaryPort();
6        pbc.bypassExpansionPort ();
7        for (int i = 0; i < NUM_DISKS; i++) pbc.bypassDisk(i);
```

```
8
9       fc.Loopinitialize();
10
11      fc.send_LIPF7F7();
12      FCstatus fs = receiveLIPStatus();
13      if (fs! = LIP_RECEIVED) error (ENCLOSURE_FAULT);
14
15      for (i = 0; i < NUM_DISKS; i++)
16      {
17          if (pbc.diskInstalled(i))
18          {
19              pbc.unBypassDisk(i);
20              fc.send_LIPF7F7();
21              fs = receiveLIPStatus();
22              if (fs! = LIP_RECEIVED)
23              {
24                  pbc.bypassDisk(i);
25                  error(DISK_FAILURE);
26              }
27          }
28      }
29
30      for (i = 0; i< NUM_DISKS; i++)
31      {
32          if (pbc.diskInstalled(i))
33          {
34              ses.issueTestUnitReady();
35              SES status ss = ses.receiveSESStatus(buffer);
36              IF (ss! = GOOD) error (TEST_UNIT_READY_FAILURE);
37              ses.issueInquiry();
38              SS = ses.receiveSESStatus(buffer);
39              if (ss! = GOOD) error (INQUIRY_FAILURE);
40              else fc.report(buffer);
41              ses.issueReadCapacity();
42              SS = ses.receive SESStatus(buffer);
43              if (SS! = GOOD) error (READ_CAPACITY_FAILURE);
44              else report (buffer);
45          }
46      }
47
48      if (pbc.signal Detect Primary ()) pbc.unBypass PrimaryPort();
49      if (pbc.signal Detect Expansion()) pbc.unBypass ExpansionPort();
50  }
```

The routine "powerUp" takes instances of the controller PBC, FC, and SES classes as arguments pbc, fc, and ses. As noted above, these class instances represent functionalities available to the STMP. These functionalities

may be spread over a number of different hardware and software components, and are described, in general, in previous subsections. The integer buffer "buffer," declared above on line 3, is local storage for information solicited from disk drives by certain SES commands. On lines 5 and 6, powerUp bypasses both the primary

5   and extension ports. On line 7, powerUp bypasses each of the disk drives within the HAE. Upon the completion of line 7, all disks in the HAE are bypassed and the FC arbitrated loop implemented on an LCC is bypassed both at the primary GBIC and the extension GBIC in order to create an FC arbitrated loop internal to the HAE. On line 9, powerUp initializes the internal FC arbitrated loop. On

10   line 11, powerUp sends a loop initialization primitive to the internal FC arbitrated loop. On line 12, powerUp waits to receive the loop initialization primitive from the FC arbitrated loop that was sent on line 11. If the status returned by member function "receiveLIPStatus," called on line 12, is not LIP_RECEIVED, as detected by powerUp on line 13, powerUp calls the error routine with the error

15   ENCLOSURE_FAULT. This error corresponds to a non-functional FC arbitrated loop on the HAE in which powerUp is being run. Otherwise, on lines 15-28, powerUp one-by-one configures on disk drives into the FC arbitrated loop and tests to see that they respond to a loop initialization primitive. For each disk, powerUp determines, on line 17, whether the disk is installed. If so, then on line 19,

20   powerUp configures the disk into the FC arbitrated loop and, on line 20, sends out a loop initialization primitive. On line 21, powerUp calls the member function "receiveLIPStatus" to wait for the loop initialization primitive sent on line 20 to be returned. If receiveLIPStatus does not return a status of LIP_RECEIVED, as detected by powerUp on line 22, then powerUp infers that the disk drive configured

25   into the FC arbitrated loop on line 19 is defective, bypasses the disk on line 24, and raises a DISK_FAILURE error on line 26. Once all the disks have been configured into the FC arbitrated loop, at the conclusion of the -for- loop comprising lines 15-28, powerUp executes a second for- loop, comprising lines 30-46, in which each disk is further tested. For each disk, powerUp issues a

TEST_UNIT_READY SES command on line 34 and receives the response to the TESTUNITREADY command on line 35. If the returned SES status is not GOOD, then powerUp calls the error routine with an indication of a TEST_UNIT_READY_FAILURE. Otherwise, on line 37, powerUp issues an

5 INQUIRY SES command to the disk currently being tested. If the return status is not GOOD, then powerUp calls the error routine, on line 39, with an indication of an INQUIRY_FAILURE error. Otherwise, on line 40, powerUp calls the member function "report" to return to the host computer or to place into an internal table information returned by the disk drive in response to an INQUIRY SES command.

10 Similarly, on lines 41-44, powerUp issues to the disk being tested a READ_CAPACITY SES command and either calls the error routine with a READ_CAPACITY error indication if the READ_CAPACITY fails, or reports the results of READ_CAPACITY_FAILURE SES command either to the host computer or stores the results in internal tables. When all disk drives have been

15 tested in the for - comprising lines 30-46, powerUp removes the bypass from the primary and expansion ports on lines 48 and 49. At this point, the powerUp routine is completed and the HAE is fully incorporated into the FC arbitrated loop. Note that, an instance of the FC class "sc" may incorporate initialization and configuration of both FC arbitrated loops within the HAE, or, conversely,

20 powerUp routines may run on both LCCs of a HAE in order to fully self-test the HAE.

The separate interrupt service routine "LIP_F8_ISR" responds to interrupts generated within a microprocessor of an LCC in response to receipt by the L_Port corresponding to the microprocessor, i.e. the enclosure services L_Port,

25 when the enclosure services L_Port receives a loop initialization primitive, described above, that indicates that an L_Port on the FC arbitrated loop has detected port failure of an upstream L_Port.

```
1       void LIP_F8_ISR (int alpa)
2       {
3         int upstream Alpa;
4
5         upstream Alpa = fc.getUpstreamAlpa (alpa);
6         if (in (&OurApas, upstreamAlpa))
7             pbc.bypassDisk(fc.getDiskNo (upstreamAlpa));
8         return  // return from interrupt;
9       }
```

The LIP_F8_ISR receives the AL_PA of the L_Port that is reporting failure in an upstream device. On line 5, LIP_F8_ISR calls the member function "getUpstreamAlpa" in order the determine the AL_PA of the upstream defective L_Port. Then, on line 6, LIP_F8_ISR determines, by calling the member function "in" whether the indicated upstream L_Port is within the HAE. If so, then on line 7, LI_F8_ISR calls the PBC controller member function "bypassDisk" to bypass the defective disk.

Thus, the above-provided implementation describes an embodiment with an arbitrated loop self-test management routine run on one or both processors within a HAE to verify operation of the HAE prior to configuration of the HAE into a series of HAEs connected to a host computer. When the HAEs are self-testing, unreliable or faulty HAEs are prevented from being configured into an FC arbitrated loop, thus increasing the overall reliability and availability of the host computer/HAEs system.

Although the present invention has been described in terms of a particular embodiment, it is not intended that the invention be limited to this embodiment. Modifications within the spirit of the invention will be apparent to those skilled in the art. For example, the present invention may be practiced in multi-peripheral-device enclosures that use different inter and intra-enclosure communications media than the FC communications medium employed in the above-described embodiment. As another example, in number of different types of controllers, microprocessors, and port bypass circuits can be used in any number of different configurations to provide the three-tiered port bypass circuit control strategy of the

present invention. Additional redundancies in controllers, microprocessors, communications busses, and firmware and software routines can be employed to further increase reliability of a multi-peripheral-device enclosure designed according to the method of the present invention. The power-up self-test routine

5   can be implemented in any number of different computer languages in a practically limitless number of ways using different modularization, control statements, variables, and other programming devices, in different sequences. Different strategies may be incorporated in the self-test routine and in other software and firmware routines running on host computers and multi-peripheral-device enclosure

10   processors for handling defective or malfunctioning components. Different components may be tested and isolated, if necessary, from operation of the remaining components.

The foregoing description, for purposes of explanation, used specific nomenclature to provide a thorough understanding of the invention. However, it

15   will be apparent to one skilled in the art that the specific details are not required in order to practice the invention. In other instances, well-known circuits and devices are shown in block diagram form in order to avoid unnecessary distraction from the underlying invention. Thus, the foregoing descriptions of specific embodiments of the present invention are presented for purposes of illustration and description; they

20   are not intended to be exhaustive or to limit the invention to the precise forms disclosed, obviously many modifications and variations are possible in view of the above teachings. The embodiments were chosen and described in order to best explain the principles of the invention and its practical applications and to thereby enable others skilled in the art to best utilize the invention and various embodiments

25   with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents: